GQL: A Graphical Query Interface for Relational Databases

Malathy Naguleswaran
&
Neville Churcher

Technical Report COSC 7/89

GQL: A Graphical Query Interface for Relational Databases

MALATHY NAGULESWARAN AND NEVILLE CHURCHER

Department of Computer Science, University of Canterbury, Private Bag, Christchurch, New Zealand.

Correspondence regarding this paper should be sent to:

Dr. N. I. Churcher
Department of Computer Science
University of Canterbury
Private Bag
Christchurch
NEW ZEALAND

electronic mail:
neville@cantuar.uucp
...!mcvax!cantuar!neville

Telephone: +64 3 642-353
Fax: +64 3 642-999
Telex: UNICANT NZ4144

SUMMARY

The linear data manipulation languages (DMLs) used to query typical relational databases have a number of practical disadvantages for users of all levels. The aim of graphical query languages is to provide efficiency for the expert and ease of use for the novice, without losing the power of a traditional DML. GQL provides a graphical query interface for relational databases. Queries are expressed graphically and translated into a DML before being passed to the host database for execution. The graphical interface of the Macintosh is used to provide facilities for incremental editing of queries as well as the display of results. A local dictionary is used to store metadata, which may be extracted from the host database or defined within GQL, for use in query formulation. Predefined relationships (or "join recipes") may be used to construct linkage conditions in queries. Parametrized subqueries may be stored for reuse, enabling both the provision of "black box" query components for use by novices and the maintenance of libraries of frequently-used components by experts.

KEY WORDS Graphical query languages Graphical interfaces Database management

INTRODUCTION

A typical modern computer system has a database management system (DBMS) as a major component. The DBMS will almost certainly be based on the relational model of data[1]. Such systems are being applied to an increasing variety of problem areas, and must accommodate users with differing levels of expertise.

A DBMS user communicates with the database by means of a query language, or data manipulation language (DML). These languages have evolved to match the advances in DBMS hardware and software[2]. Users expect the ability to pose unrestricted *ad hoc* queries. Popular query languages such as QUEL[3] and SQL[4] are based on the relational algebra or relational calculus. Although such languages have many desirable properties, such as relational completeness[5], they have some practical disadvantages.

A typical DML is intimidating and confusing to a novice user. Extensive knowledge of the underlying schema of the database may be required in order to form correct and meaningful queries. This frequently leads to the construction of a shell to insulate the user from the DML. However this has the disadvantage of restricting the user's ability to pose *ad hoc* queries.

Users can suffer impaired productivity because of the difficulty of correctly representing a complex query within the syntax of a DML, where all query components are visible simultaneously. Even in cases where the DML has some "structure", it is difficult to distinguish between particularly important query components, such as linkage conditions, and those which are less significant, such as selection conditions involving constants. These and other features of typical DMLs make it difficult to re-use query fragments and to apply structured programming concepts to query formation.

The advantages of diagrammatic techniques have long been recognized, particularly in systems analysis and design applications. The view that "a picture is worth a thousand words" is applicable to many aspect of languages or interfaces, including the process of query formation[6,7]. A number of systems, with varying degrees of graphical content, have been proposed as alternatives to traditional linear DMLs. These range from tabular systems to completely pictorial or iconic systems. Some are based on specific database concepts such as the entity-relationship (ER) model[8] and the universal relation[9]. Such systems are becoming increasingly relevant as workstations with high quality graphics capabilities become more affordable, replacing the expensive special-purpose hardware previously required.

We have implemented GQL[10], a graphical query language interface for relational databases. The Apple Macintosh™ was chosen as an example of a low-cost, widely available workstation which is capable of supporting graphical query languages. The user incrementally constructs a graphical query, which is then translated into QUEL and passed to a host DBMS for execution. No modifications to the host DBMS are required. GQL can thus be used as a front end to any QUEL-speaking DBMS. Alternatively, its output can be passed through some suitable filter such as a QUEL to SQL translator[11].

Our objective is to use the graphical interface of the Macintosh to provide a structured representation of queries which reduces the complexity of the query formulation process. We aim to show that "a structured picture is worth more than a thousand words." Only components "relevant" to a particular level should be visible at the same time. Irrelevant details may be collapsed into structured query components. The internal structure of a query component is revealed by exploding it to reveal a further level of details which naturally belong together. At

each step in the query construction process the user chooses from a finite number of valid operations, thus reducing the complexity of the process.

A local dictionary contains metadata extracted from the system relations of the host DBMS as well as GQL-specific metadata such as the definitions of parametrized subqueries. The GQL user may thus be easily restricted to relevant parts of the database without the necessity for elaborate security mechanisms. Although GQL is not strictly based on the ER model, the concept of "join recipes", which are equivalent to relationships, is used. These may be stored in the dictionary or defined by the user.

The resulting system is suitable for use by a variety of users. Beginners may only make use of predefined queries initially, progressing to customising of these and the definition of new ones as confidence increases. They are shielded from the syntactic details of the DML into which their queries will ultimately be translated. The ability to view simultaneously the query, in its graphical form, and the results is valuable for teaching purposes. Experienced users benefit from the power of the graphical interface, which deals in semantic units rather than characters, and from the ability to re-use complex query components.

Only enquiries are supported by the current GQL implementation. Queries involving creation, modification or deletion of data require a fuller knowledge of the database schema than is assumed by GQL. For example, integrity constraints could be violated if the user is unaware of the existence of the attributes and relations involved. This does not mean that graphical interfaces are unsuited to non-enquiry queries. A more sophisticated interface between the dictionaries GQL and the DBMS is required.

QUERIES AND QUERY LANGUAGES

A query has two major components: a *target list* and a *qualification.* The target list specifies the data items to be returned by the query. These may be relations, attributes or aggregate functions, such as counts and averages. Several relations may be involved, and a particular relation may appear more than once (see figure 3). The qualification, which may consist of a number of clauses, states conditions which must be met by tuples satisfying the query. Such conditions may involve attributes not included in the target list.

A suitable framework for discussing queries and query languages is provided by the relational calculus, a form of first-order predicate calculus, or the relational algebra to which it is equivalent[5,9]. In calculus terms, the target list involves attributes specified by a number of tuple variables, while the qualification is a predicate in which the tuple variables will usually also appear. In algebraic terms, the qualification may involve the selection and join operators, while projection of the required attributes is implicit in the target list.

The QUEL language, into which GQL translates graphically formed queries, was chosen because it bears a strong resemblance to the relational calculus and is thus a good intermediate form for further translation to other query languages such as SQL. Many query languages have been developed, mostly based on the relational calculus formalism.

Graphical query languages must provide many of the features of linear query languages, such as:

- Explicit or implicit definition of tuple variables and specification of target list attributes. These correspond to "entity instances".

- Linkage conditions, which relate attribute values of one tuple variable to corresponding values of another. These correspond to "relationships" between the entities.
- Selection conditions, involving attribute values corresponding to a single tuple variable.
- Aggregate functions such as counting the number, or finding the mean of values satisfying some condition.

Aspects of query formation which users find most difficult are the correct specification of linkage conditions, particularly when more than two relations are involved, and the correct use of multiple tuple variables ranging over the same relation. These two problems frequently occur together. The first situation arises where the conceptual data model contains n-ary relationships, while the second results from the presence of self-relationships. GQL simplifies these problematic activities by providing an environment which reduces the complexity of query construction tasks while providing powerful facilities.

Other graphical query interfaces have been proposed. Some, like Query By Example[12], are tabular, rather than graphical in nature. Others are iconic, where each possible query is activated by an interface symbol. A third category is pictorial where parts of a bit-mapped graphical picture are "hot-spots" leading to query actions. These categories will not be treated further here, as their objectives are different from ours.

More relevant to our considerations are general graphical query systems. These include CUPID[13] which is based on QUEL, PICASSO[14] which is based on the universal relation interface and gql/ER[15] based on the ER model. The PICASSO project, while more sophisticated than GQL, is not as generally applicable as it is implemented on top of a specific DBMS. Like GQL, PICASSO is a retrieval-only

system. Some systems, such as CUPID, provide a graphical representation of the required query components but do not employ any structuring to hide irrelevant query details. The graphical representation is "flat", and the complexity of the graphical query as perceived by the user may be as high, or higher than that of its equivalent in a linear DML.

Systems based on the ER model may require more system support and restrict the queries available to the user. The relational model allows joins (≅ relationships) to be performed on any two attributes with a common domain. Most implementations further extend this to include attributes whose domains are implemented as compatible system types. The concept of an extra layer of predefined relationships is useful, but it is important in a query interface to be able to assert the presence of new relationships dynamically, without having to go through a data modelling tool first. Such relationships may be only temporary, or relevant to a single user, and not to be incorporated in the overall conceptual data model. GQL allows the use of predefined linkage conditions, as well as other types of subquery, but also allows the freedom to define new ones during the enquiry session.

GQL OVERVIEW

The GQL environment is intended to encourage top-down development of queries. The GQL interface is shown in figure 1. A *palette* of *tools* is visible at the left of the query window, while further options are available as *items* of the *menus* in the *menu bar* at the top of the window. The tools are used for common query editing operations such as creating, linking, selecting, moving, deleting, exploding or collapsing query components. The menus contain additional functions such as relationship definition and the area at the bottom of the screen is used to display

linkage conditions. The remainder of the screen is used for query construction, which may involve opening other windows. Further details are given in Naguleswaran[10].

When the query is complete it is passed to the DBMS for execution. The results are displayed in a separate window and may also be saved in a file. The results may be displayed together with the corresponding query, as in figure 1. It is also possible to view the QUEL form of the translated query. Both of these features are useful for checking a complex query or where GQL is being used as a teaching aid.

The tuple variables participating in a query are shown as rectangles with rounded corners containing the corresponding variable names. Multiple tuple variables from the same relation are manipulated separately, emphasizing the distinctions between them. Attribute level activities, such as selecting the target list or specifying aggregate operations, typically involve "exploding" the symbols corresponding to tuple variables. The employee symbol in figure 1 has been exploded, and the attributes number and name have been selected for inclusion in the target list. Attributes may be removed from the display with the deletion tool without affecting the underlying database schema.

Some restrictions on the layout of the query have been imposed. These are intended to encourage the formation of left-to-right readable queries. Readability is important as there is a considerable amount of text in a GQL query and simplicity of layout aids clarity. We aim to keep the number of simultaneously visible query components within the "magic number" 7 ± 2[16]. This is reflected in the number of available tools and in the number of query components which fit comfortably onto the screen at once. If the screen becomes crowded then subqueries may be used or some exploded items may be collapsed or moved. We

believe that the solution to such problems is not simply to scroll a flat unstructured graphical query or to display it on a larger screen. This results in the loss of some advantages of the graphical interface without necessarily reducing the perceived complexity of query construction.

Queries involving more than one tuple variable require at least one linkage condition in order to be meaningful in GQL. The user may access these in a number of ways. If the two "ends" already exist then the "connection" tool may be used to establish a link between them by allowing the user to choose from a list of predefined named join conditions. If only one end exists then the connection tool allows the user to choose from a list containing relationship names and the relations to which their other ends are connected; this process is shown in figure 2. When the manages employee relationship is selected a second employee tuple variable and the appropriate relationship connection appear (see figure 3).

Tuple variables from the same relation are distinguished by unique numbers. Relationship connections are shown as lines connecting the tuple variable symbols, each has a node symbol containing the corresponding relationship text. Choosing from a list of valid possibilities is a less complex task than the usual situation where the user must suggest valid linkage conditions herself. If the required linkage condition is not available then it may be created and added to the dictionary for subsequent re-use.

Simple selection conditions involving constants are placed immediately below the attribute in the exploded view. The two employee tuple variables in figure 3 have been exploded, revealing that the name and salary attributes of both are in the target list and that simple selection conditions have been placed on the number attributes of both. The area at the bottom of the query window is used to display more complex expressions in selection conditions or join definitions. Such

conditions often represent relationships which are not already in the GQL dictionary. For example, the condition in figure 3 might represent a paid less than relationship. The expression tool is used to construct expressions by selecting appropriate attributes and comparison operators. Many conditions may be specified for a given query.

The use of subqueries is one of GQL's most important features. Any GQL query may be named and stored for subsequent re-use. This provides a mechanism for making predefined queries available to novice users. The query of figure 3, in which an employee both manages and is paid less than another, could be saved as a subquery underpaid. It could then be retrieved, possibly modified and re-used as before.

Another important use of subqueries is as components in a larger query. The tuple variables of a stored subquery function as parameters which must be linked to the appropriate tuple variables of the main query. The subquery is displayed at half normal size to allow the connections to be made, as in figure 4, and may then be collapsed to hide its internal details. A query may involve many subqueries, which may in turn contain further subqueries, though directly or indirectly recursive definitions are forbidden.

IMPLEMENTATION

GQL runs on Apple Macintosh computers with at least 512k memory and is supported by a data dictionary. This is stored locally as only a small part of the dictionary contents refers directly to the schema of the DBMS and the retrieval-oriented nature of GQL suggests that such references should be read-only. Changes to the DBMS schema should be made with an appropriate data modelling

tool and propagated to the GQL dictionaries. Almost all of GQL's processing is carried out on the workstation without requiring constant communication with the host DBMS. In fact for many purposes, such as teaching QUEL or query predefinition there is no need to communicate with the host at all.

The dictionary contains three layers of information. GQL is written in Pascal and the dictionary is implemented as a number of Pascal record types. However these are normalized so that all or part of the dictionary could be easily moved to the host DBMS.

The first layer contains the basic information extracted from the system relations of the host DBMS. This includes such data as relation names together with the names and domains of their attributes. This is the information which is certain to be available in any host system. The second layer adds semantic information for use in query construction. In particular the join recipes, or relationships, are stored at this level. Composite attributes are required for cases where the join condition involves multiple attributes; this corresponds to a composite foreign key. The third layer holds subqueries for re-use. The structure of the dictionary is given in Naguleswaran[10].

The process of translating the GQL query into QUEL involves a number of steps:

- declaration of tuple variables
- construction of target list
- construction of selection and linkage conditions
- assembling complete query
- checking query validity.

This process is complicated by the fact that subqueries may be involved. It has been necessary to restrict the functionality of GQL to ensure that the operators

provided are sufficiently generic that they may be supported by reasonably direct translation into typical DMLs such as QUEL. This is particularly relevant in the case of aggregate operations.

The resulting QUEL query is available for inspection and is then transmitted to the host. The current communication mechanism uses the serial port of the Macintosh to communicate with an Ingres[3] database running under Unix™. The query is passed to the monitor and is processed by the DBMS in the usual way. The output is returned via GQL and is displayed in a separate window (figure 1). The host DBMS handles matters such as query optimization. It would also be possible to communicate with a purpose-built monitor to achieve finer control, such as tuple-at-a-time retrieval.

CONCLUSIONS

A graphical query interface for relational databases has been implemented on the Apple Macintosh™. Facilities, including re-usable components, have been provided for incrementally constructing and editing queries, which are then translated into a form suitable for processing by a host DBMS. The interface is intended to reduce the perceived complexity of the query by suppressing details which logically belong at lower levels. The resulting system is suitable for novice users, who will make extensive use of "canned" queries, as well as for expert users. GQL illustrates the ability of widely available graphical workstations to function as interfaces to mainframe software such as a DBMS. Other activities such as data modelling and schema maintenance can also benefit from a similar approach.

REFERENCES

1. E. F. Codd, 'A Relational Model for Large Shared Data Banks', *Comm. ACM,* **13**, (6), 377-387, (1970).

2. M. Jarke and Y. Vassiliou, 'Frame Work for Choosing a Database Query Language', *ACM Comp. Surv,.***17**, (3), 313-340, (1985).

3. M. Stonebraker, E. Wong, P. Kreps and G. Held, 'The Design and Implementation of INGRES', *ACM TODS.* **1**, (3), 189-222, (1976).

4. M. M. Astrahan and D. D. Chamberlin, 'Implementation of a Structured English Query Language' *Comm. ACM* **18**, (10), 580-587, (1975).

5. C. J. Date, *An Introduction to Database Systems Vol. 1 4ed.* Addison-Wesley, 1986.

6. S.-K. Chang, 'Visual Languages: A Tutorial and Survey', in *Visualization in Programming, Lecture Notes in Computer Science vol 282,* eds. P. Gorny and M. J. Tauber, Springer-Verlag, 1986.

7. S.-K. Chang, T. Ichikawa and P. A. Ligomenides, (eds.) *Visual Languages,* Plenum, 1986.

8. P. P.-S. Chen, 'The Entity-Relationship Model - Toward a Unified View of Data', *ACM TODS,* **1**, (1), 9-36,.(1976).

9. J. D. Ullman, *Principles of Database Systems 2nd Edition.* Computer Science Press, Rockville, MD, 1982.

10. M. Naguleswaran, 'A Graphical Relational Query Language on Apple Macintosh', *M. Sc. Thesis*, University of Canterbury, 1988.

11. J. Webb, 'A QUEL to SQL Data Manipulation Language Translator', *Project Report*, Computer Science Dept., University of Canterbury, 1988.

12. M., M. Zloof, 'Query-by-Example: A Data Base Language', *IBM Systems Journal.*, **16**, (4), 324-343, (1977).

13. N. McDonald, and M. Stonebraker, 'CUPID - the Friendly Query Language', in *DATA: its Use, Organization and Management - ACM Pacific Conference*, 1975.

14. H.-J. Kim, H. F. Korth and A. Silberschatz, 'PICASSO: A Graphical Query Language', *Software—Practice and Experience*, **18**, (3), 169-203, (1988).

15. Z.-Q. Zhang and A. O. Mendelzon, 'A Graphical Query Language for Entity-Relationship Databases', in *Entity-Relationship Approach to Software Engineering*, eds. C.G. Davis, S. Jajodia, P. A. Ng, and R. T. Yeh, North-Holland, 1983.

16. G. A. Miller, 'The magical number 7 plus or minus two: Some limits on our capacity for processing information', *Psychological Review* **63**, 81-97, (1956).

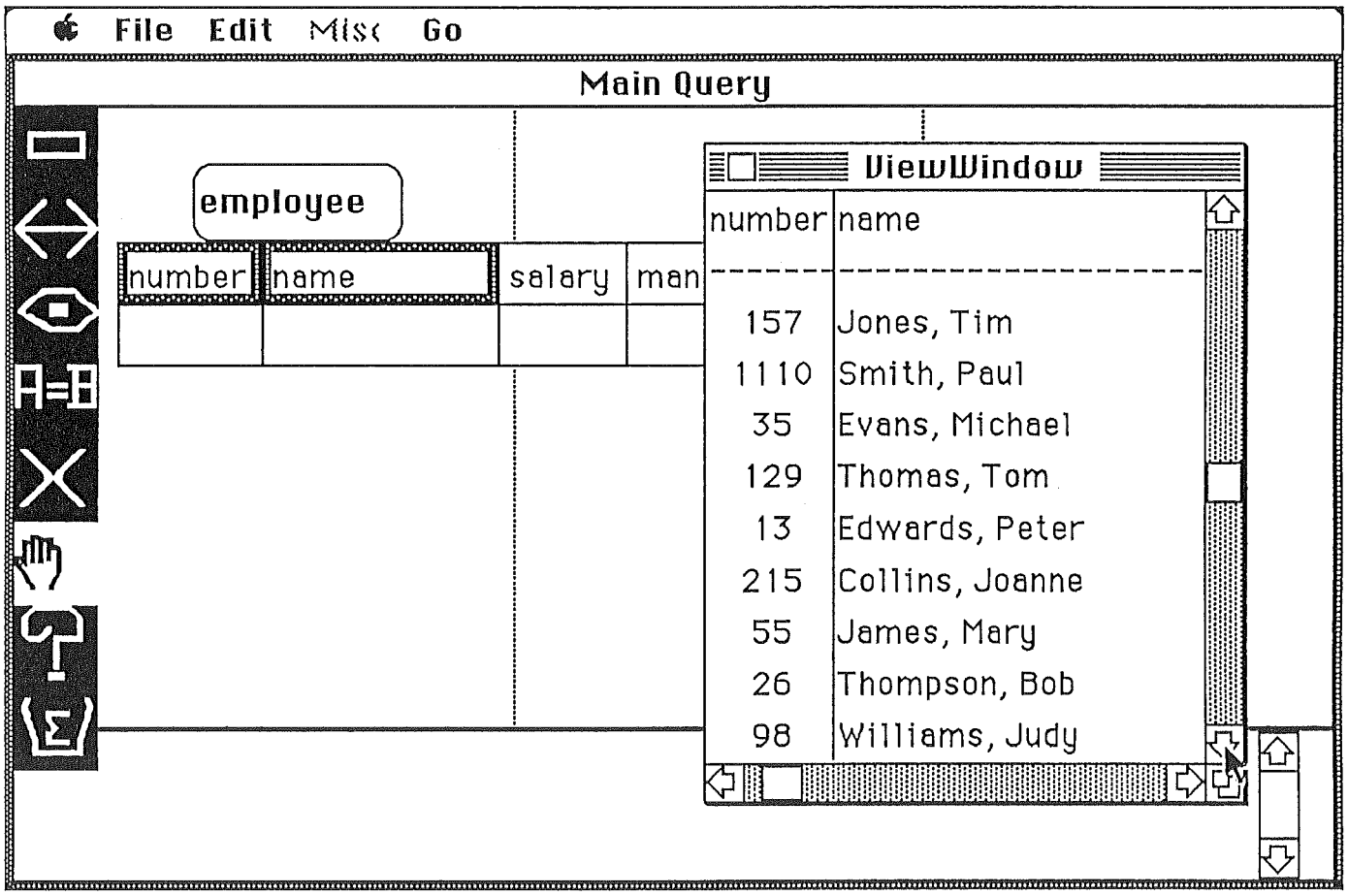
FIGURE CAPTIONS

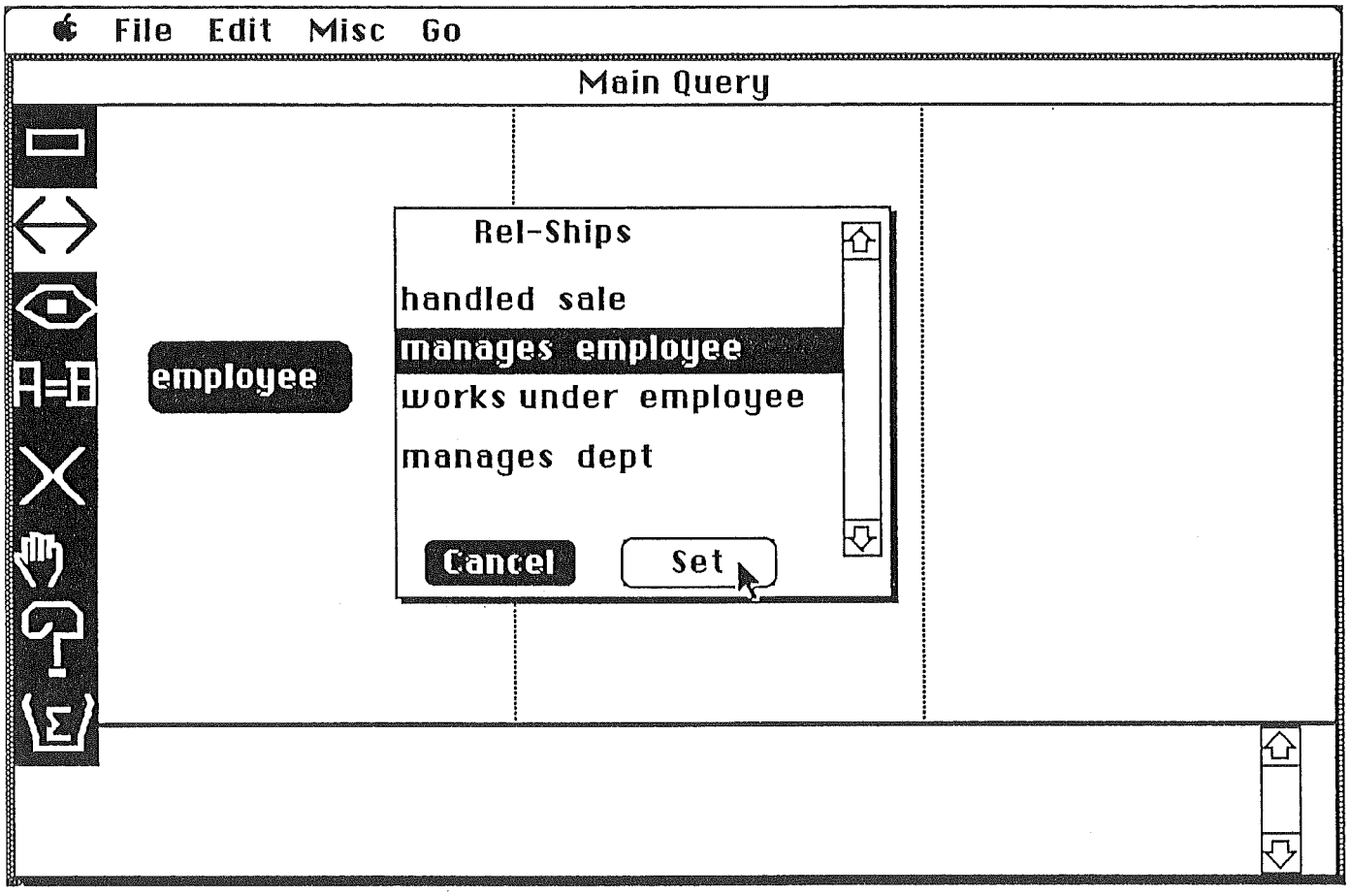
Figure 1: The GQL environment. The query shown is "*list the numbers and names of all employees.*"

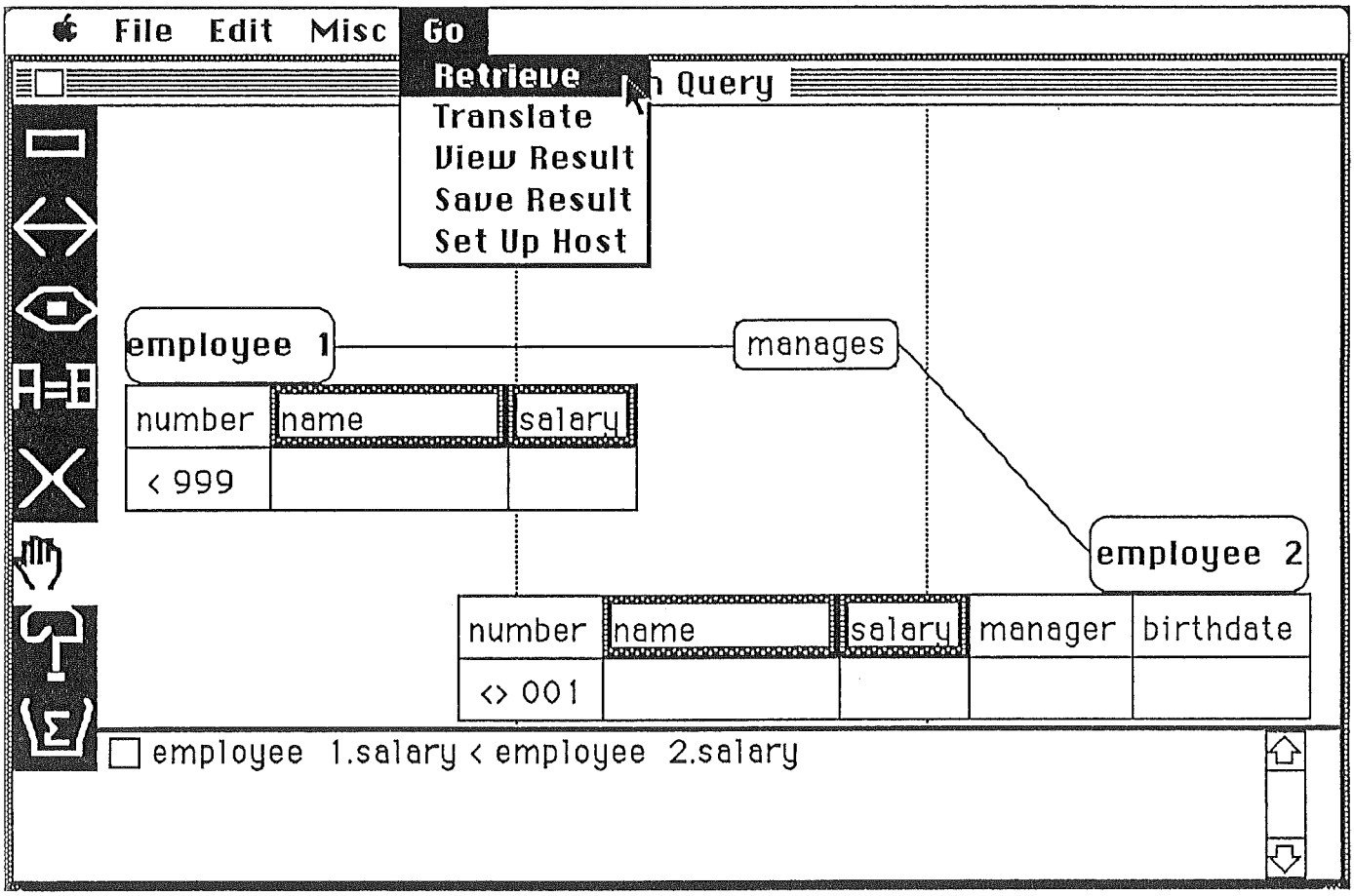
Figure 2: Using the connection tool. Selection is made from a list of relationships in which an employee may participate.

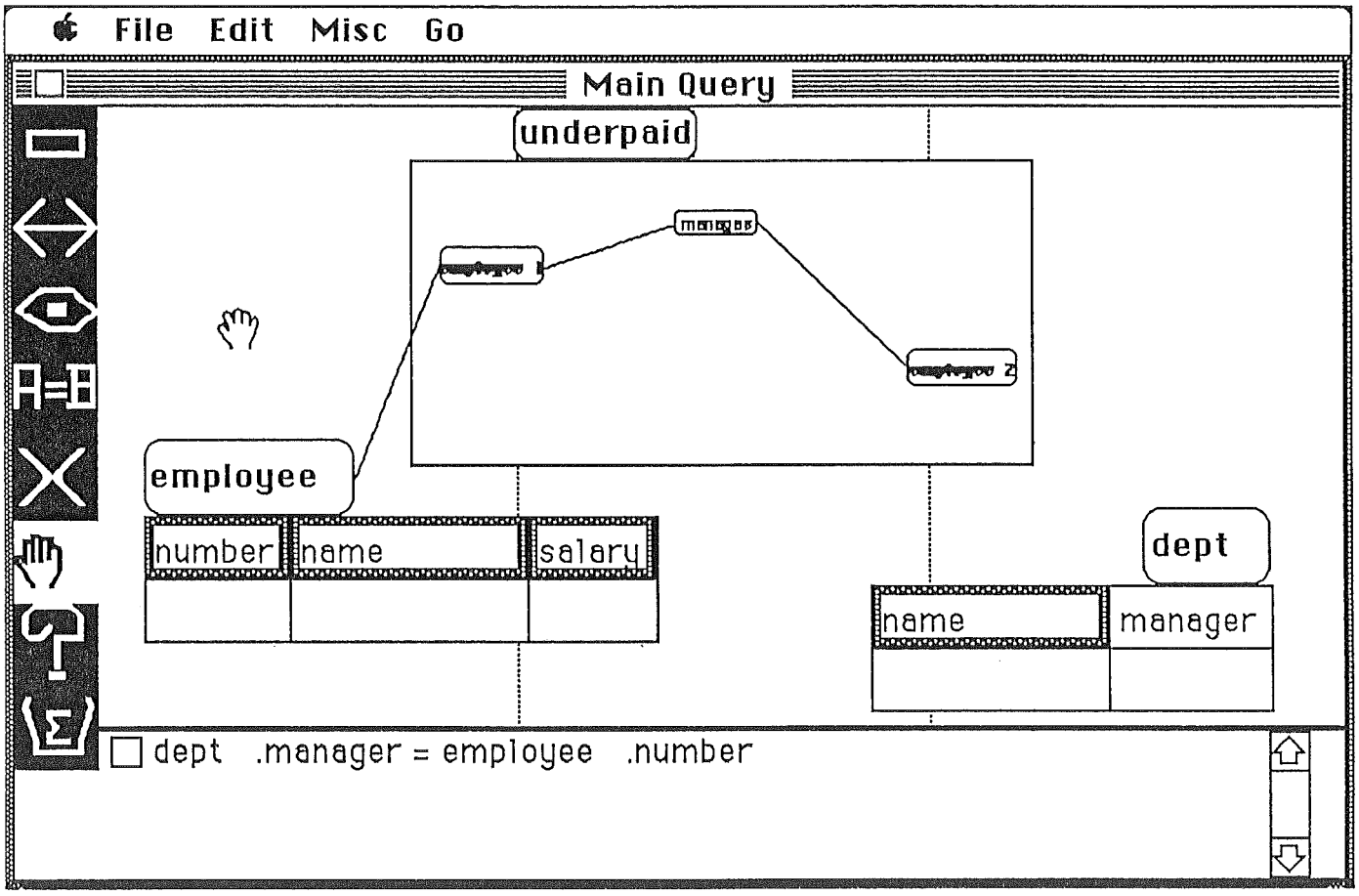
Figure 3: A more complicated query involving a predefined relationship, two simple selection conditions and a linkage condition. The query represented is "*If any employees, with the exception of employee number 001, are paid more than their managers, if the managers employee number is less than 999, then retrieve the name and salary of both employee and manager.*"

Figure 4: Using a subquery. The subquery "underpaid" consists of the query components of figure 3. The query shown is "*if there are any departments whose managers are underpaid then retrieve the department name, and the number, name and salary of the employee who manages it.*"


File Edit Misc Go
Main Query
employee
number
name
salary
man
ViewWindow
number
name
157 Jones, Tim
1110 Smith, Paul
35 Evans, Michael
129 Thomas, Tom
13 Edwards, Peter
215 Collins, Joanne
55 James, Mary
26 Thompson, Bob
98 Williams, Judy


File Edit Misc Go
Main Query
Rel-Ships
handled sale
manages employee
works under employee
manages dept
employee
Cancel
Set


File
Edit
Misc
Go
Retrieve
Translate
View Result
Save Result
Set Up Host
Query
employee 1
manages
number
name
salary
< 999
employee 2
number
name
salary
manager
birthdate
<> 001
employee 1.salary < employee 2.salary


File Edit Misc Go
Main Query
underpaid
employee
number
name
salary
dept
name
manager
dept .manager = employee .number